5. 条件分岐と繰り返し 1

Conditional Branching and Repetition 1

プログラミング・データサイエンス I

2021/5/20

1　今日の目的

今日の目的

- ここまでの例は、開始から終了まで一直線の処理
- 実際のプログラムでは
  - 条件にあうところだけ実行
  - 操作を繰り返す
  - 異常なことが起こった時の対処

ここまでの例では、プログラムの最初から最後へ向かって、基本的に、一直線に処理が進んでいきました。

実際のプログラムでは、条件にあう部分だけを実行する、ある処理を繰り返す（これは、すこし出てきましたね)、異常が起こったら対応する、などが必要になります。

さて、今日のテーマに関連するサンプルプログラムをダウンロードしてください。ダウンロードの仕方は、2 回目の講義で説明しました。忘れてしまった人は、参照してください。

`https://github.com/first-programming-saga/Control`

2 インデントとプログラムブロック：Indents and Program Blocks

インデントとプログラムブロック：Indents and Program Blocks

- Python では、インデントを使って、プログラムブロックを区別
  - 条件を満たした場合に実行する部分
  - 繰り返し部分
- プログラムブロック：プログラムの塊

条件分岐や繰り返しを記述するためには、条件を満たしたら実行する部分、繰り返し実行する部分を指定する方法が必要になります。python は他のプログラミング言語とは非常に異なった方法で、プログラムの一部分を指定します。

インデント (indent) とは、行の先頭にある空白のことです。日本語でも英語でも、段落の先頭は一文字開けますね。これもインデントです。インデントすることで、視覚の上で、区切りやまとまりを示すことができます。python では、このインデントに特別な意味を与えます。そのため、不用意に空白を行の先頭に入れてはいけないのです。

python では、インデントを使って条件を満たしたら実行する部分、繰り返し実行する部分を指定します。この部分のことを、「プログラムブロック (program block)」と呼びます。

3 条件分岐：Conditional Branching

条件分岐：Conditional Branching

- 条件に応じて、処理内容を分ける
  - 条件を満たす場合：`True`
  - 条件を満たさない場合：`False`

今日は、条件に応じて処理内容を分ける「条件分岐」について主に説明します。

条件とは、比較 (ある変数がある値より大きいなど) などを行った結果です。比較などの結果は `True` か `False` という値であることは、以前に説明しました。

3.1 条件分岐:1

条件分岐:1

- 条件を満たす場合の処理を指定
- インデントの使い方に注意

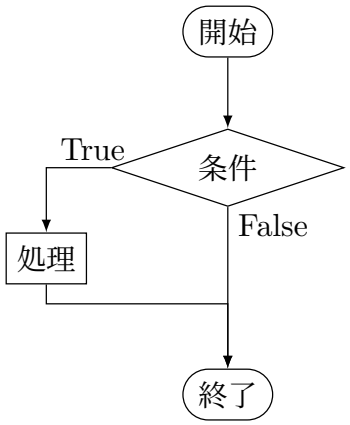


図 1　条件分岐の例 1。条件を満たす場合の処理を指定

すごく簡単な例を示します。図 1 は流れ図 (flow chart) という処理の流れを図示したものです。ひし形の部分に条件を書き、四角で処理を表します。今回は、条件を満たす場合には、なにかの処理を行い、満たさない場合には、何もしないという例です。

ソースコード 3.1　簡単な条件分岐

```
1 a = −10
2 if a < 0 :#a が負ならば、符号を変える
3     a *= −1
4 print(a)
```

if.ipynb を開いてください (ソースコード 3.1)。最初のセルのプログラムを見てください。2 行目で a という変数が負であることを調べています。この条件を満たしたときは、3 行目を実行します。3 行目の先頭にスペースが 4 つあります。これがインデントです。

もう一つ注意してください。2 行目の条件の後ろに:があります (#から後ろはコメント)。これがプログラムブロックの開始を表しています。なお、VSCode では、:を入力して改行すると、自動的にインデントしてくれます。

このプログラムの動作は予想できますか? a という変数が正ならば何もせず、負ならば符号を変え、値を印刷します。つまり、必ず正の値を印刷します。

3.2 条件分岐:2

条件分岐:2

- 条件を満たさない場合の処理も指定
- else の使い方

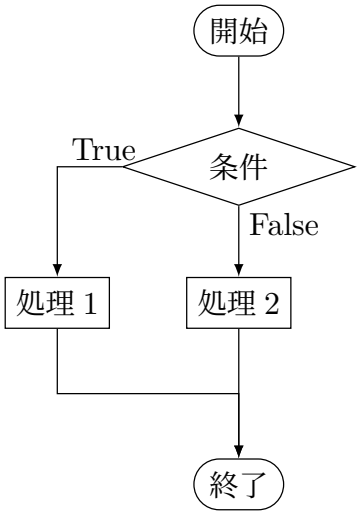


図 2　条件分岐の例 2。条件を満たさない場合の処理も指定

ソースコード 3.2　簡単な条件分岐 2

```
1 a = −20
2 if a < 0 :#a が負の場合
3     b = −a
4 else :#a が負でない場合
5     b = a
6 print(b)
```

次は、条件を満たした場合と満たさない場合にそれぞれに処理がある場合です (図 2)。if.ipynb の二番目のセルです。4 行目の else:で、「条件を満たさない場合」を指定しています。

このプログラムの結果は、前のシートのものと同じです。違いは、今回のものでは変数 a の値を直接印刷しているのではなく、別の変数 b に代入した後に印刷している点です。

3.3 条件分岐:3

条件分岐:3

- 条件に論理演算がある場合
- and、or、not
- 適切に () を使う

ソースコード 3.3　論理演算を含む条件分岐 1

```
1 message = '未設定'
2 if a > 0 and b > 0 :
3     message = 'a␣もb␣も正です'
4 if a > 0 or b > 0 :
5     message = 'a␣かb␣の少なくとも一方が正です'
6 print(message)
```

ソースコード 3.4　論理演算を含む条件分岐 2

```
1 message = '未設定'
2 if (a > 0 and b > 0) and c > 0 :
3     message = 'a␣もb␣もc も正です'
4 if (a > 0 or b > 0) and c > 0 :
5     message = 'a␣かb␣の少なくとも一方が正で、c は正です'
6 print(message)
```

if.ipynb の 3 番目から 5 番目のセルでは、条件文に and や or が入っています。このように複数の条件を論理演算で結ぶことも可能です。

5 番目のセルのように三つ以上の条件を書く場合には、どの部分を先に判定するかを示す () を使います。() を書かないと、結合の強いものから順に評価をします。論理演算では、not が最も結合が強く、次が and、最も弱いのが or です。これを覚えておくのは大変ですから、() を使って、はっきりと書きましょう。

3.4 条件分岐:4

条件分岐:4

- 条件文の入れ子構造
- elif の使い方

ソースコード 3.5　elseif の例

```
1  if a == b : #a と b が等しい場合
2      c = a
3      message = "同点"
4  elif a > b:
5      c = a
6      message = f'a={c}␣b={b}␣でa␣の勝ち'
7  else:
8      c = b
9      message =f'a={c}␣b={b}␣でb␣の勝ち'
10 print(message)
```

条件の中に、さらに条件がある場合があります。if.ipynb の 6 番目と 7 番目のセルを見てください。

a と b が異なる場合、つまり 4 行目の else の中で、更に a と b の大小関係を調べています。6 行目と 7 行目、9 行目と 10 行目のインデントがスペース 8 個になっています。これは、4 行目の else のさらに内側の 5 行目の if、8 行目の else の部分に対するプログラムブロックであることを示しています。

同じことを別の形式で表したものが、ソースコード 3.5 です。4 行目の elif は else if の略で、1 行目の条件を満たさなかった場合に、更に条件を判定することを表しています。

課題 1　$x < 0$ の場合は $y = -1$、$0 \leq x < 1$ の場合は $y = 0$、それ以外は $y = 1$ となるプログラムを作成しなさい。また、その動作を確認しなさい。

補足です。他のプログラミング言語には、場合分けをする switch のような構文を持つものがあります。しかし、python にはそれはありません。ここの例のように elif を使って書くことになります。

4　while 文：while loops

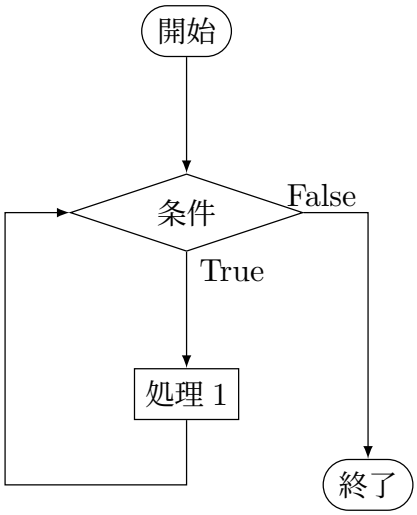
条件を満たす限り繰り返す:Indefinite iterations

- while の使用
- 繰り返し回数に注意

今日は、条件を使う構文をもう一つ説明します。「条件を満たす限り繰り返す」という構文です。英語の文を見ると、回数を限定しない繰り返しという意味になっています。

図 3 を見てください。条件を満たすと「処理」を行い、再び条件の部分に戻ります。条

図 3　条件を満たす限り繰り返す



件を満たさなくなると、終了します。

例は、`while0.ipynb` の最初のセルです (ソースコード 4.1)。初めに $c = 0$ とします。while ループの中で c を印刷するとともに、1 だけ増やします。やがて $c = 10$ となると条件を満たさなくなり、終了します。

ソースコード 4.1　簡単な `while` 文

```
1 c = 0
2 while c < 10:
3     print(c)
4     c += 1
```

二番目のセルの例をソースコード 4.2 に示します。

`print(f 'c = {c}, s = {s}')` の部分を説明しておきましょう。`f` は format の意味で、「書式」つまり印刷形式を指定します。`{c}`には、変数 `c` の値が、`{s}`には変数 `s` の値が入ります。実際に動かすと、どのような出力になるかが、すぐにわかります。

もう一つ注意しておきたいことがあります。`while` は条件を満たしている限り繰り返します。つまり、いつまでも終了しない「暴走」状態になる危険性があります。そのため、`while` を使う際には、停止することを走らせる前に確認しましょう。それでも暴走してしまった場合には、メニュー中の赤い四角の部分を押します (図 4)。

ソースコード 4.2　簡単な `while` 文:2

```
1 s = 0
2 c = 0
3 while s < 100:
```

```
4       s += c
5       print(f'c␣=␣{c},␣s␣=␣{s}')
6       c += 1
7   print(s)
```

図 4　プログラムを強制停止するには、メニュー中の赤い四角の部分を押す

5　while を使った例：Examples of while loops

5.1　Euclid 互除法: Euclidean Algorithm

Euclid 互除法: Euclidean Algorithm

- 二つの自然数の最大公約数を求める
- while の例

ソースコード 5.1　Euclid 互除法

```
1   n=465
2   m=360
3   if m > n :#m>n の場合は入れ替え得る
4       x = n
5       n = m
6       m = x
7
8   while m > 0 :
9       r = n % m
10      print(f'{n}␣%␣{m}␣=␣{r}')
11      n = m
12      m = r
13  print(n)
```

while の例として、二つの自然数の最大公約数を求める、Euclid の互除法を示します。高校の 1 年で習いましたか？

二つの自然数 m と n を与えます。n のほうが大きいとします。$m > 0$ である限り、以下を繰り返します。

1. n を m で割った余りを r とする
2. n に m を代入する
3. m に r を代入する

最後に得られた n の値が最大公約数となります。

プログラムは、ソースコード 5.1 です。

5.2 break と continue

break と continue

- `while` ループの途中から抜ける方法
- break の使い方
- continue の使い方

`break` と `continue` について説明します (ソースコード 5.2)。`while.ipynb` を開けてください。少し難しいかもしれません。また、これらの方法を使わなくても同様のことはできます。そのため、「こういうこともできる」という程度の理解で十分です。

ソースコード 5.2　乱数を生成し、場合分けをする例

```
1 while True:
2     x = randint(0,99)
3     if x % 5==0:
4         break#while ループを抜ける
5     elif x % 7 == 0:
6         continue#while ループの先頭へ
7     print (x)
8 print("end with "+str(x))
```

繰り返し処理を行うプログラムでは、途中で繰り返しを止める、あるいはプログラムブロックの残りを実行せずに繰り返しの次の番に移りたいことがあります。

例を見てください。`while` ループは条件に `True` とありますから、無限ループです。その次の行で、x には、0 から 99 までのでたらめな整数が入ります。x が 5 で割り切れたら `while` ループを抜けるというのが `break` です。x が 7 で割り切れたら、`while` の先頭に

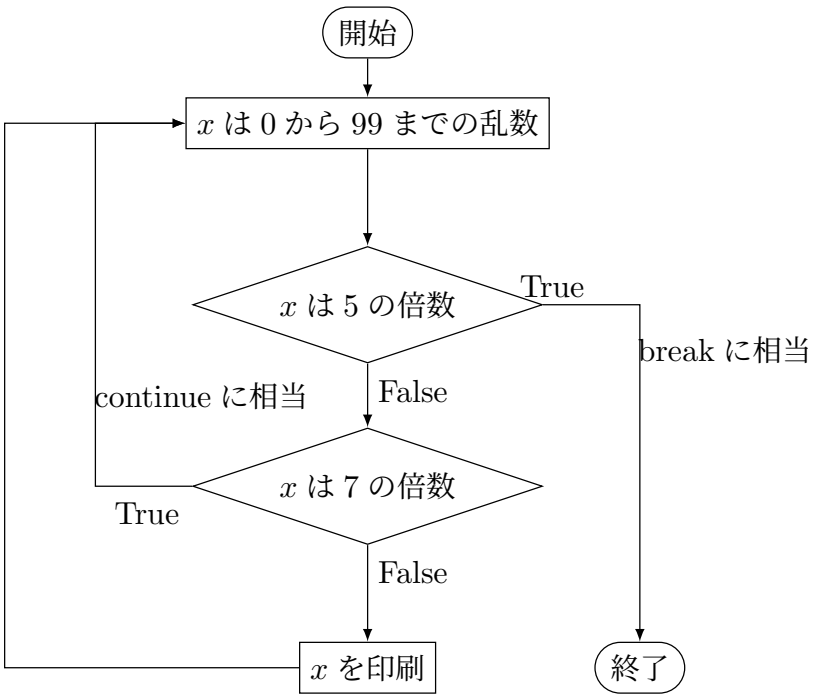


図 5　乱数を生成し、場合分けをする例

戻るというのが `continue` です。流れ図を図 5 に示します。

5.3　while と else

while と else

- `while` ループが中断しなかった場合の処理
- 素数か否かの判断の例

`break` は `while` ループを中断して外へ出てしまいます。中断せずに `while` ループの最後まで至った時の処理を記載するのが、`while` の後の `else` です。

ソースコード 5.3 は、自然数 n が素数かを判定するプログラムです。n は偶数ではないとします。ある奇数が素数かを判定するのは、非常に手間がかかります。その方法は、3 から始めて n の平方根まで順に奇数で割ってみるしかありません。途中で割り切れたら、素数ではないことが分かります。割り切れなかったら素数となります。つまり `while` ループが完了すると `else` の部分に至って、素数であるというメッセージを出します。手順をフローチャートでも示しています (図 6)。

ソースコード 5.3　素数か否かの判定

```
n = 219
if n <= 0:
    print(f'{n}は正でなければならない')
elif n < 2:
    print(f'{n}は素数ではない')
elif n % 2 == 0:
    print(f'{n}は偶数であり、素数ではない')
else:
    m=int(math.sqrt(n))
    k = 3
    while k <= m:
        if n % k == 0:
            print(f'{n}は{k}で割り切れるため、素数ではない')
            break#while ループから抜け出す
        k += 2
    else:#while ループの最後まで至った場合
        print(f'{n}は素数である')
```

課題 2　`while.ipynb` の末尾にある課題です。0 から 10 までの和を `while` を使って計算しなさい。無限ループにならないように、十分に注意すること。

6　次回

次回は、5 章の後半、`for` を使った繰り返しと例外処理を扱います。

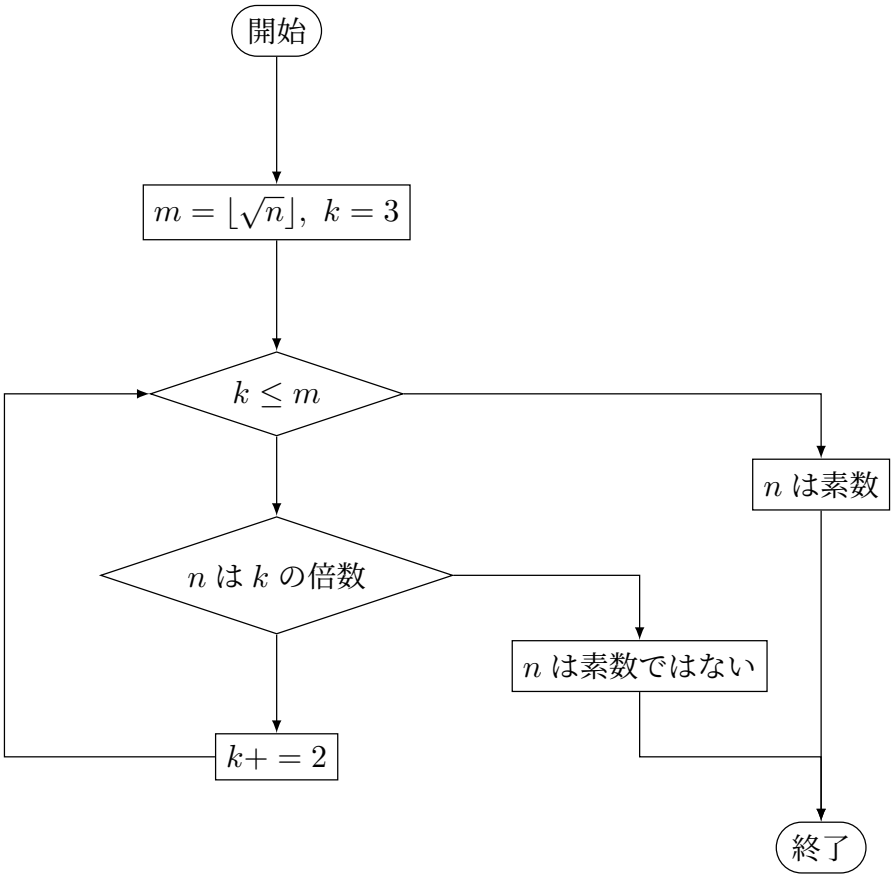


図 6　素数か否かの判定